Set-6

1. The Company ABC runs with employees with several departments. The Organization manager had all the mobile numbers of employees. Assume that you are the manager and need to verify the valid mobile numbers because there may be some invalid numbers present. Implement a LEX program to check whether the mobile number is valid or not.

   Code:

```
%{
#include <stdio.h>
%}

%%
// Valid mobile number pattern (Assuming 10-digit numbers starting with 7, 8, or 9)
[789][0-9]{9}{ printf("Valid Mobile Number: %s\n", yytext); }

// Invalid mobile numbers
[0-6][0-9]{9} | [0-9]{1,9} | [0-9]{11,}{ printf("Invalid Mobile Number: %s\n", yytext);
}

// Ignore spaces, tabs, and new lines
[ \t\n]+   ;

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  return 0;
}
```

2. Write a LEX program to find the length of the longest word.

   Code:

```
%{
```

```
#include <stdio.h>
#include <string.h>

int max_length = 0;
char longest_word[100];
%}

%%
// Match words
[A-Za-z]+ {
  int length = yyleng;
  if (length > max_length) {
    max_length = length;
    strcpy(longest_word, yytext);
  }
}

// Ignore spaces, tabs, and new lines
[ \t\n]+   ;

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  printf("Longest Word: %s (Length: %d)\n", longest_word, max_length);
  return 0;
}
```

3. Write a LEX program to print all HTML tags in the input file.

Input Source Program: (sample.html)

```
<html>
<body>
<h1>My First Heading</h1>
```

```
<p>My first paragraph.</p>
</body>
</html>
```

Code:

```
%{
#include <stdio.h>
%}

%%
// Match HTML tags
<[^>]+> { printf("HTML Tag: %s\n", yytext); }

// Ignore everything else
.  ;

%%
int main(int argc, char *argv[]) {
  FILE *file;
  if (argc > 1) {
    file = fopen(argv[1], "r");
    if (!file) {
      printf("Cannot open file %s\n", argv[1]);
      return 1;
    }
    yyin = file;
  }
  yylex();
  return 0;
}
```

4.Write a C program to construct recursive descent parsing.

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char input[100];
int pos = 0;

// Function prototypes
void E();
```

```
void T();
void F();

// Match and consume expected character
void match(char expected) {
  if (input[pos] == expected) {
    pos++;
  } else {
    printf("Error: Unexpected character '%c'\n", input[pos]);
    exit(1);
  }
}

// Grammar rules implementation
// E -> T E'
void E() {
  T();
  while (input[pos] == '+' || input[pos] == '-') {
    pos++; // Consume + or -
    T();
  }
}

// T -> F T'
void T() {
  F();
  while (input[pos] == '*' || input[pos] == '/') {
    pos++; // Consume * or /
    F();
  }
}

// F -> (E) | id (single character variables)
void F() {
  if (isalpha(input[pos])) {
    pos++; // Consume identifier
  } else if (input[pos] == '(') {
    pos++; // Consume (
    E();
    match(')'); // Expect )
  } else {
```

```
        printf("Error: Unexpected character '%c'\n", input[pos]);
        exit(1);
    }
}

int main() {
    printf("Enter an expression: ");
    scanf("%s", input);

    E();

    if (input[pos] == '\0') {
        printf("Parsing successful!\n");
    } else {
        printf("Error: Unparsed input remaining\n");
    }

    return 0;
}
```